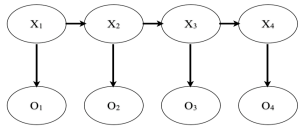
Deep Learning for NLP

CS224N

Christopher Manning

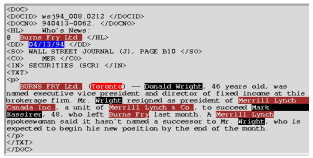
(Many slides borrowed from ACL 2012/NAACL 2013
Tutorials by me, Richard Socher and Yoshua Bengio)

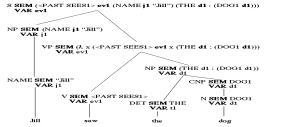
Machine Learning and NLP



NER

WordNet

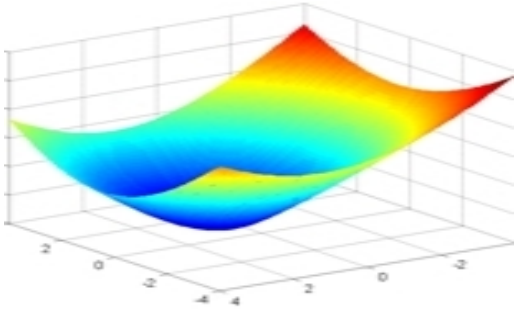




SRL Parser

Usually machine learning works well because of human-designed representations and input features

Machine learning becomes just optimizing weights to best make a final prediction



PA1 and PA3?

Representation learning attempts to automatically learn good features or representations

What is Deep Learning?

Three senses of "deep learning":

- #1 Learning a data-dependent hierarchy of multiple intermediate levels of representation of increasing abstraction
- #2 Use of distributed representations and neural net learning methods, even when the architecture isn't technically deep
- #3 Anything to do with artificial intelligence

  "Deep learning is the idea that machines could eventually learn by themselves to perform functions like speech recognition"
  http://readwrite.com/2014/01/29/

Deep Learning and its Architectures

Deep learning attempts to learn multiple levels of representation
Focus: Multi-layer neural networks

Output layer

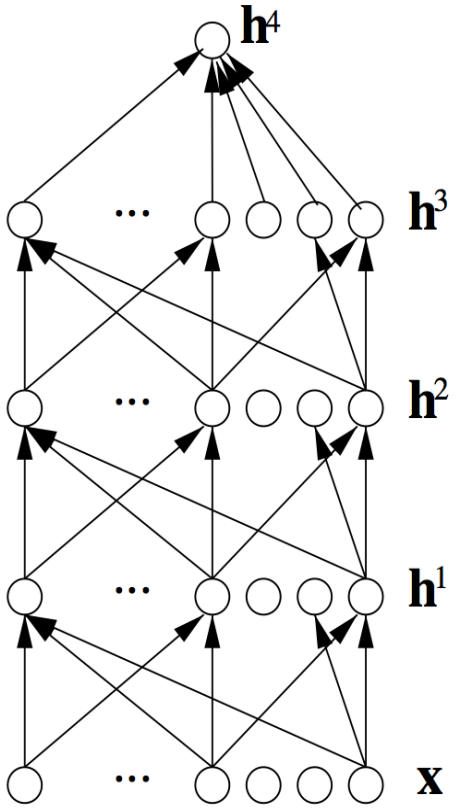
Here predicting a supervised target

Hidden layers

These learn more abstract representations as you head up

Input layer

Raw sensory inputs (roughly)



Advantages of Deep Learning (Part 1)

#1 Learning representations

Handcrafting features is time-consuming

The features are often both over-specified and incomplete

The work has to be done again for each task/domain/…

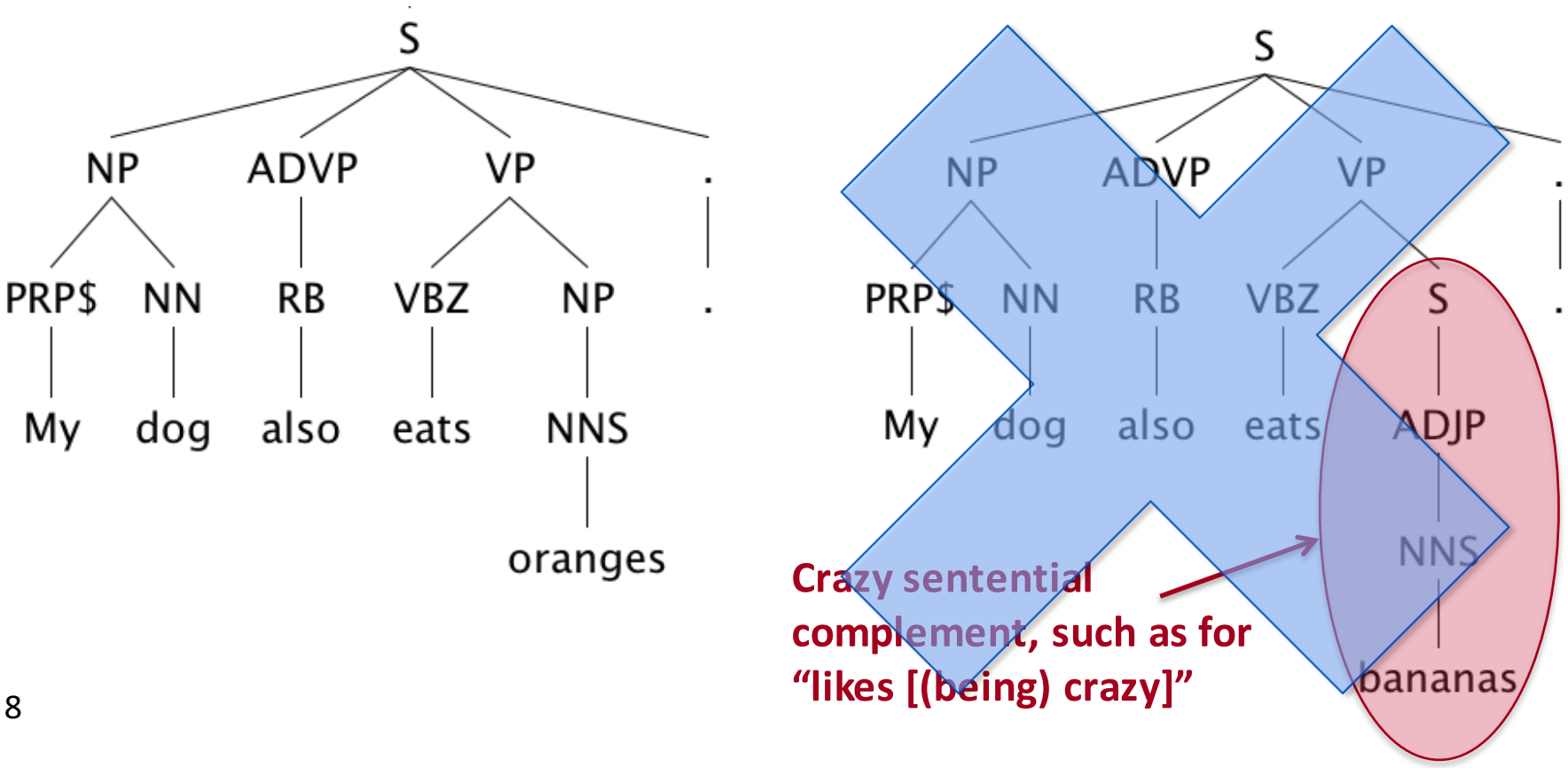
Humans develop representations for learning and reasoning

Our computers should do the same

Deep learning provides a way of doing this
Machine Learning

#2 The need for distributed representations

Current NLP systems are incredibly fragile because of their atomic symbol representations



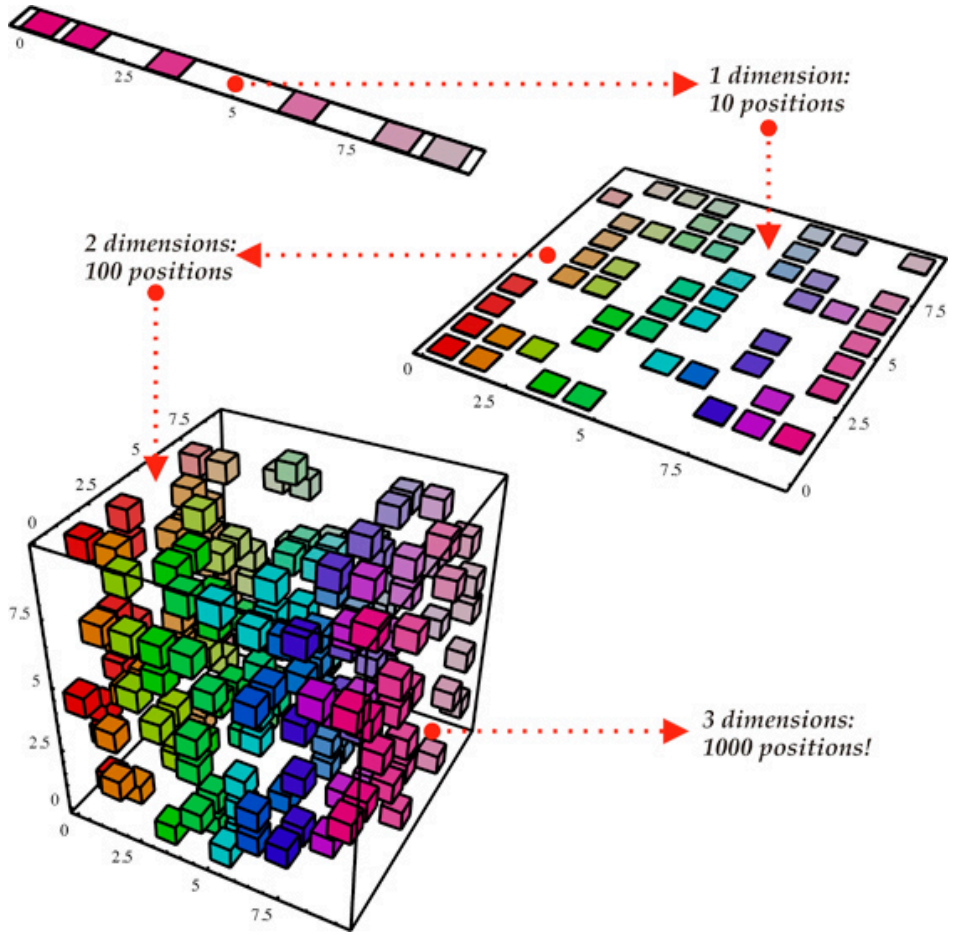
Distributed representations deal with the curse of dimensionality

Generalizing locally (e.g., nearest neighbors) requires representative examples for all relevant variations!

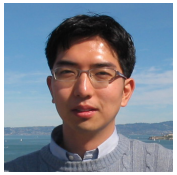
Classic solutions:

- Manual feature design
- Assuming a smooth target function (e.g., linear models)
- Kernel methods (linear in terms of kernel based on data points)

Neural networks parameterize and learn a "similarity" kernel

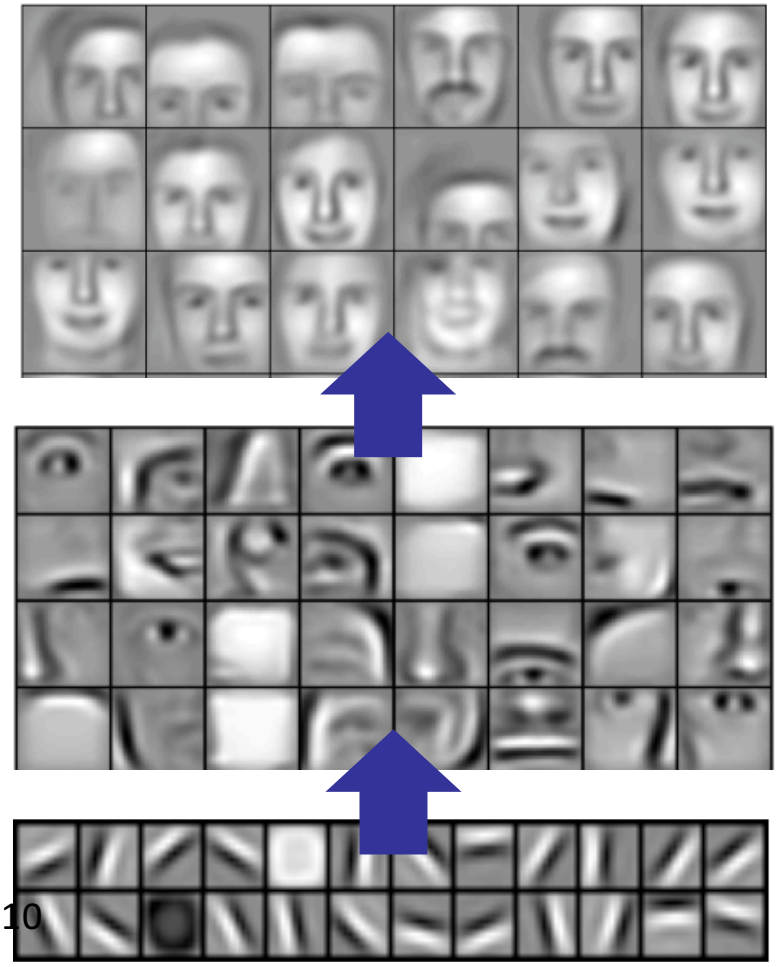


#3 Learning multiple levels of representation



[Lee et al. ICML 2009; Lee et al. NIPS 2009]

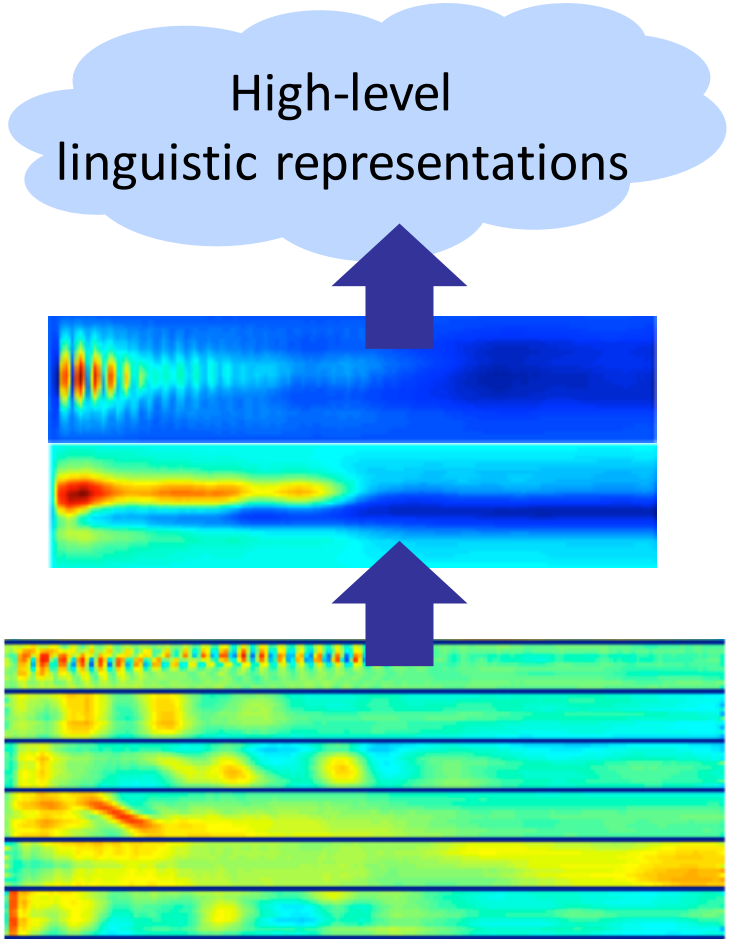
Successive model layers learn deeper intermediate representations



Layer 3

Layer 2

Layer 1



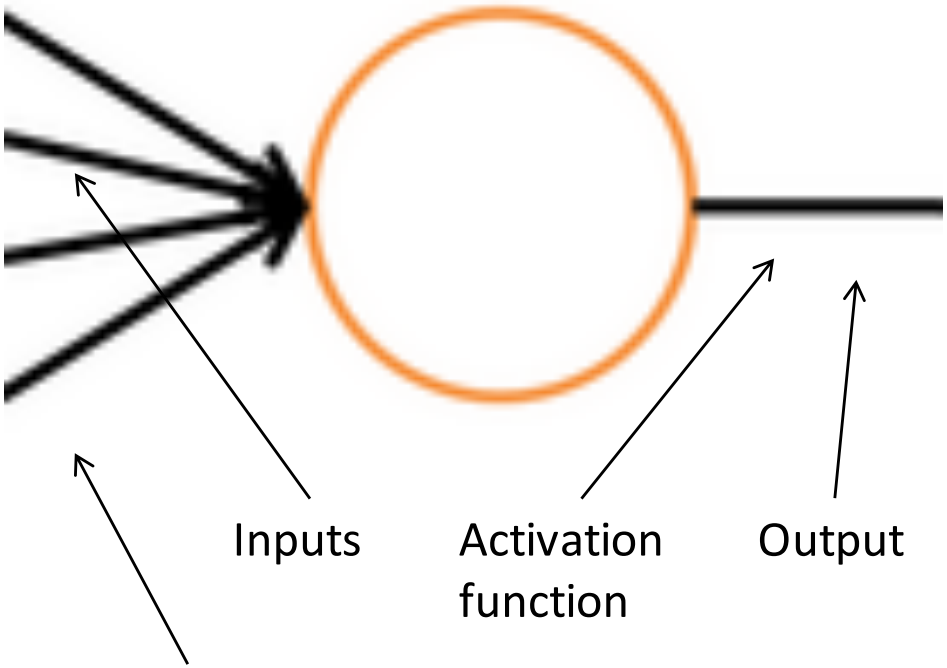
From logistic regression to neural nets

Demystifying neural networks

Terminology

Math is similar to maxent/logistic regression

A single neuron
A computational unit with *n* (3) inputs and 1 output and parameters *W, b*



From Maxent Classifiers to Neural Networks

Refresher: Maxent/softmax classifier (from last lecture)

$$P(c \mid d, \lambda) = \frac{\exp \sum_i \lambda_i f_i(c, d)}{\sum_{c' \in C} \exp \sum_i \lambda_i f_i(c', d)}$$

Supervised learning gives us a distribution for datum *d* over classes in *C*

Vector form:

$$P(c \mid d, \lambda) = \frac{e^{\lambda^\top f(c,d)}}{\sum_{c'} e^{\lambda^\top f(c',d)}}$$

Such a classifier is used as-is in a neural network ("a softmax layer")

- Often as the top layer

But for now we'll derive a two-class logistic model for one neuron

From Maxent Classifiers to Neural Networks

Vector form: $$P(c \mid d, \lambda) = \frac{e^{\lambda^\mathsf{T} f(c,d)}}{\sum_{c'} e^{\lambda^\mathsf{T} f(c',d)}}$$

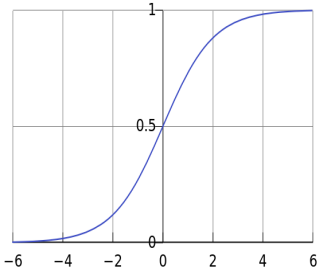
Make two class:

$$P(c_1 \mid d, \lambda) = \frac{e^{\lambda^\mathsf{T} f(c_1,d)}}{e^{\lambda^\mathsf{T} f(c_1,d)} + e^{\lambda^\mathsf{T} f(c_2,d)}} = \frac{e^{\lambda^\mathsf{T} f(c_1,d)}}{e^{\lambda^\mathsf{T} f(c_1,d)} + e^{\lambda^\mathsf{T} f(c_2,d)}} \cdot \frac{e^{-\lambda^\mathsf{T} f(c_1,d)}}{e^{-\lambda^\mathsf{T} f(c_1,d)}}$$

$$= \frac{1}{1 + e^{\lambda^\mathsf{T}[f(c_2,d) - f(c_1,d)]}} = \frac{1}{1 + e^{-\lambda^\mathsf{T} x}}$$

for $x = f(c_1,d) - f(c_2,d)$

$$= f(\lambda^\mathsf{T} x)$$



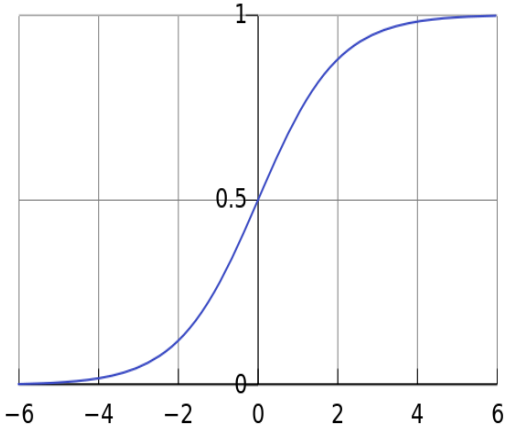
for $f(z) = 1/(1 + \exp(-z))$, the logistic function – a sigmoid non-linearity.

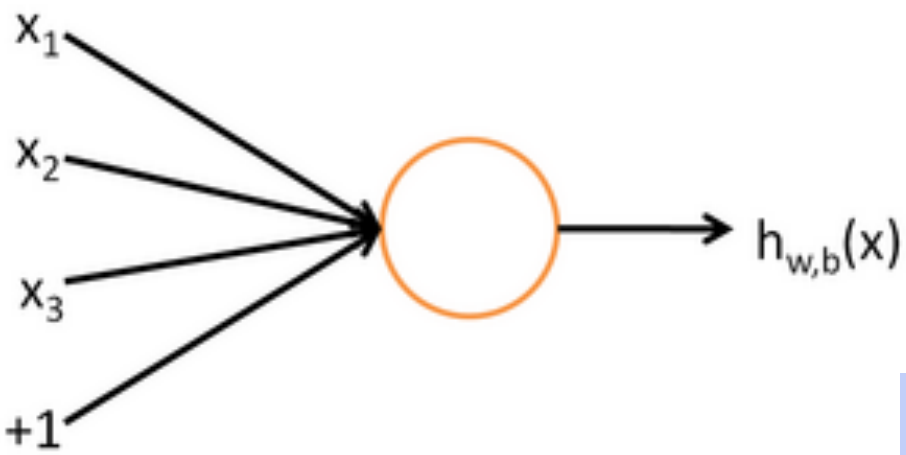
This is exactly what an artificial "neuron" computes

$$h_{w,b}(x) = f(w^{\mathsf{T}}x + b)$$

b: We can have an "always on" feature, which gives a class prior, or separate it out, as a bias term

$$f(z) = \frac{1}{1 + e^{-z}}$$

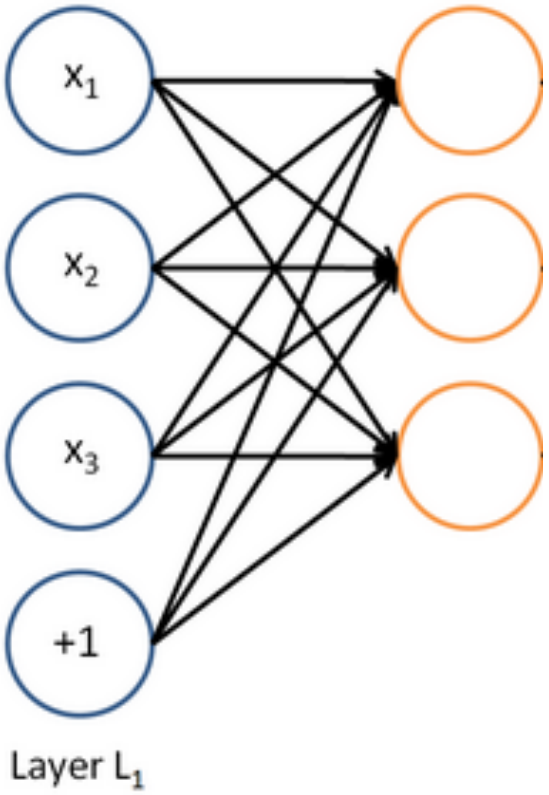




w, *b* are the parameters of this neuron i.e., this logistic regression model

A neural network = running several logistic regressions at the same time

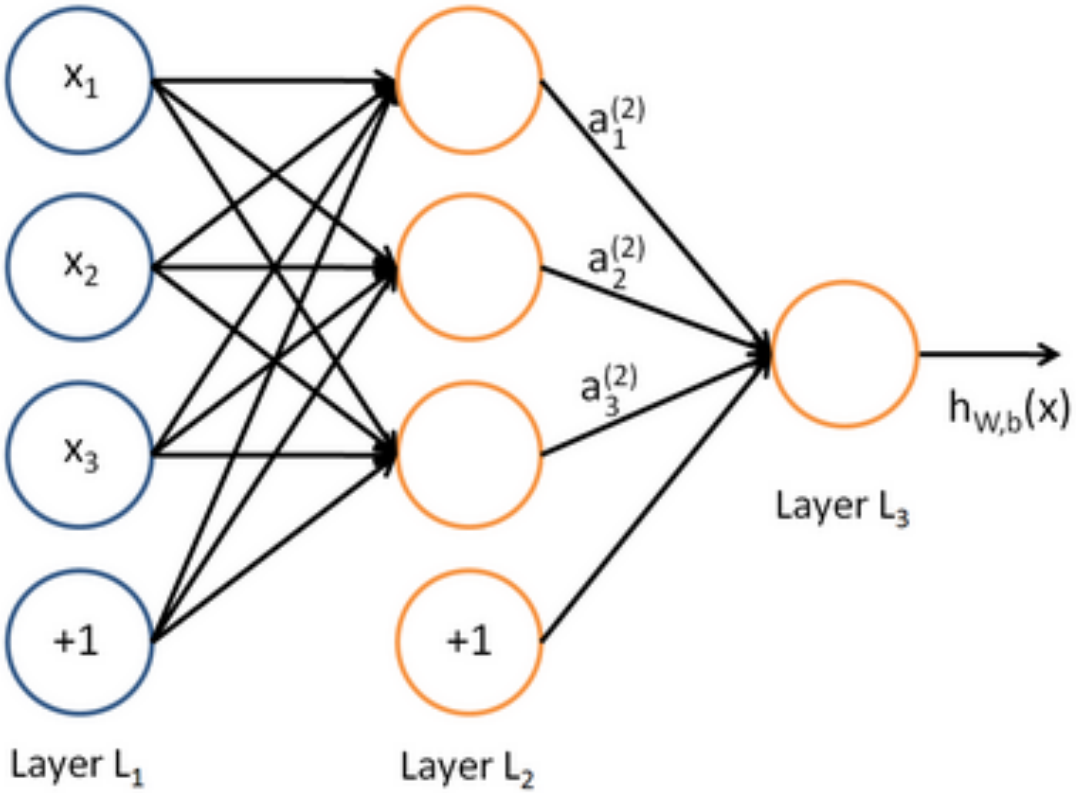
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs



But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!

A neural network = running several logistic regressions at the same time

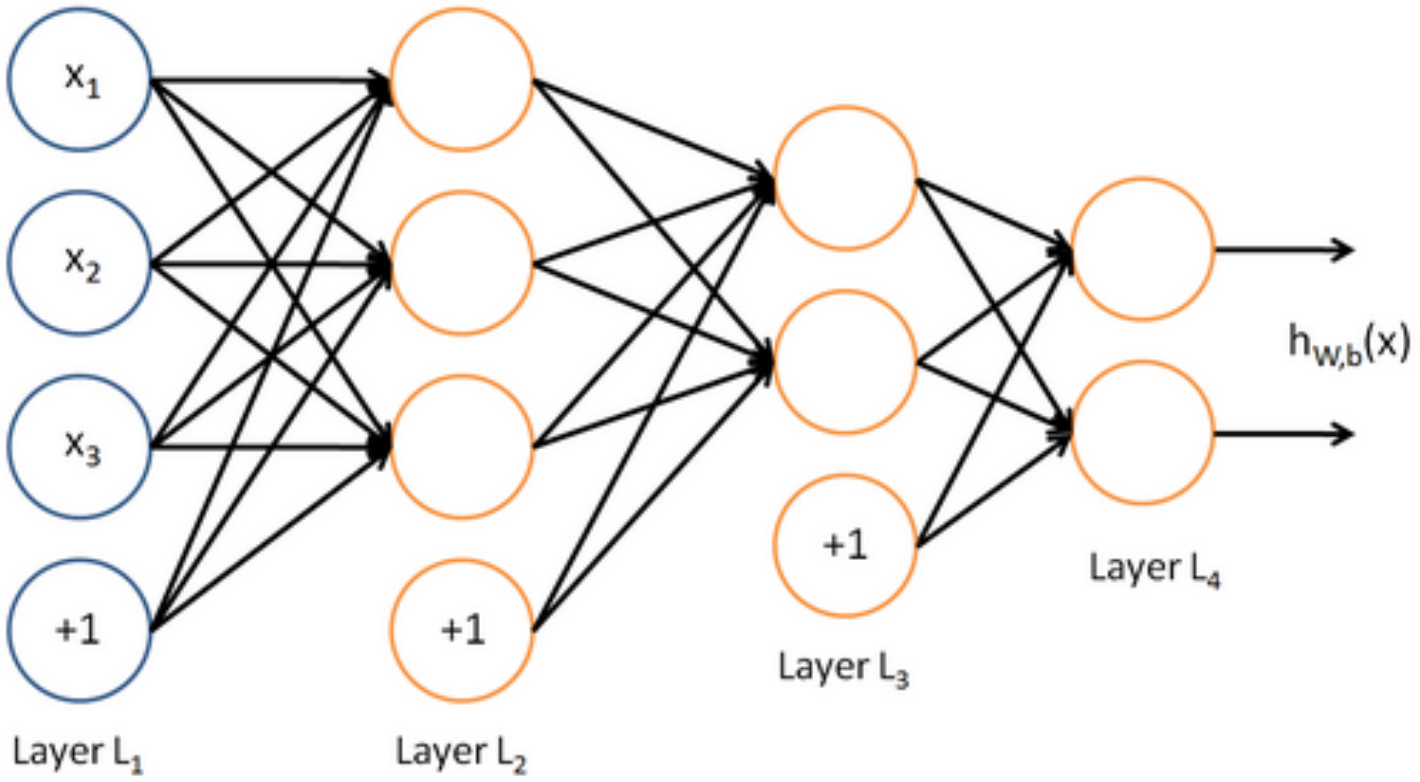
… which we can feed into another logistic regression function



and it is the training criterion that will decide what those intermediate binary variables should be, so as to do a good job of predicting the targets for the next layer, etc.

A neural network = running several logistic regressions at the same time

Before we know it, we have a multilayer neural network….



Matrix notation for a layer

We have

$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$

$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

etc.

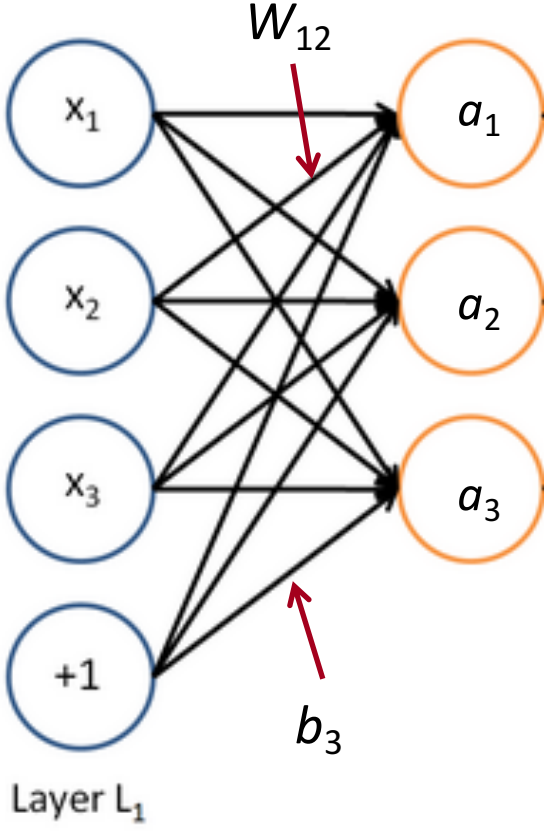
In matrix notation

$$z = Wx + b$$

$$a = f(z)$$

where f is applied element-wise:

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

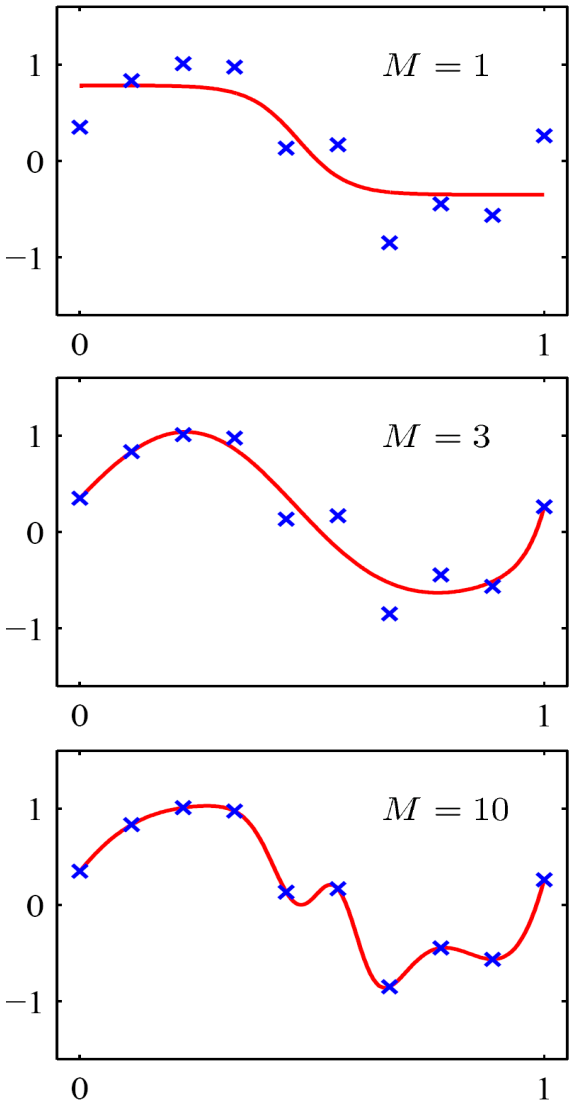


How do we train the weights W?

- For a supervised neural net, we can train the model just like a maxent model – we calculate and use gradients
  - Stochastic gradient descent (SGD)
    - Usual
  - Conjugate gradient or L-BFGS
    - Sometimes; slightly dangerous since space may not be convex
- We can use the same ideas and techniques
  - Just without guarantees …
- This leads to "backpropagation", which we cover later

Non-Linearities: Why they're needed

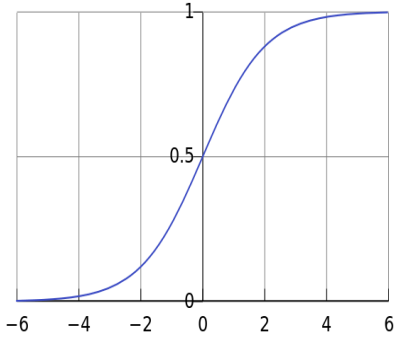
- For logistic regression: map to probabilities
- Here: function approximation,
  e.g., regression or classification
  - Without non-linearities, deep neural networks can't do anything more than a linear transform
    - Extra layers could just be compiled down into a single linear transform
- People often use other non-linearities, such as **tanh, ReLU**



Non-Linearities: What's used

logistic ("sigmoid")

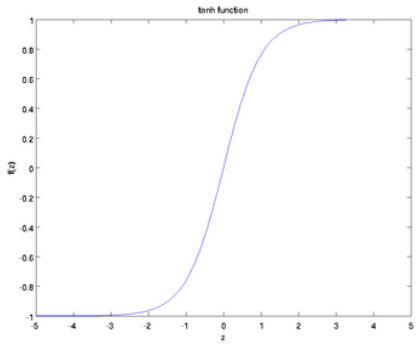
$$f(z) = \frac{1}{1 + \exp(-z)}.$$



$$f'(z) = f(z)(1 - f(z))$$

tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



$$f'(z) = 1 - f(z)^2$$

tanh is just a rescaled and shifted sigmoid $\tanh(z) = 2\text{logistic}(2z) - 1$

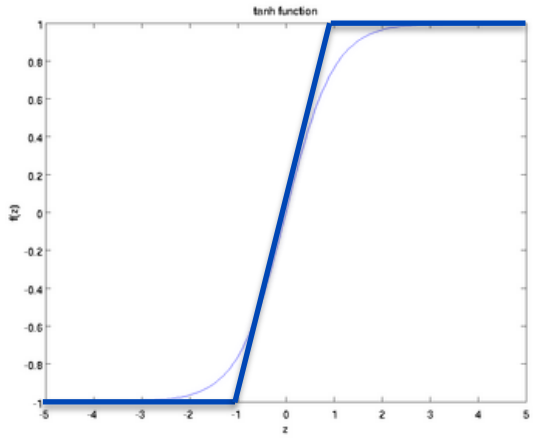
tanh is often used and often performs better for deep nets

- It's output is symmetric around 0

Non-Linearities: Other choices

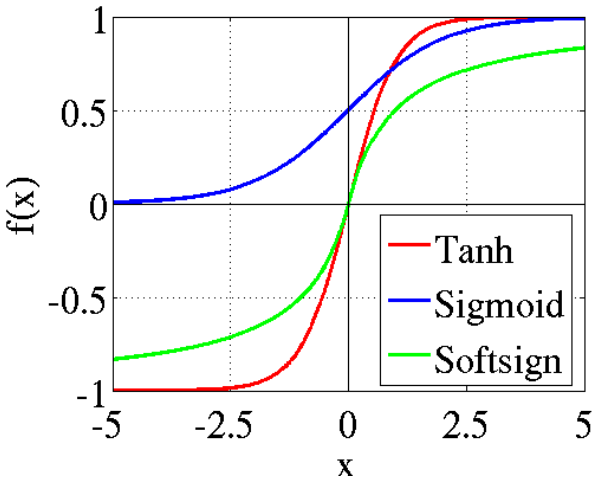
hard tanh

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases}$$



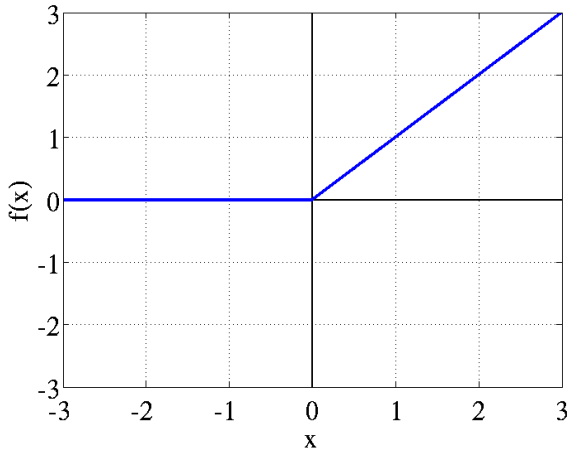
soft sign

$$\text{softsign}(z) = \frac{a}{1+|a|}$$



linear rectifier (ReLU)

$$\text{rect}(z) = \max(z, 0)$$



- hard tanh similar but computationally cheaper than tanh and saturates hard.
- [Glorot and Bengio *AISTATS* 2010, 2011] discuss softsign and rectifier
  - **Rectified linear unit** is now common – transfers linear signal if active

Summary
Knowing the meaning of words!

You now understand the basics and the relation to other models

- Neuron = logistic regression or similar function
- Input layer = input training/test vector
- Bias unit = intercept term/always on feature
- Activation = response/output
- Activation function is a logistic or other nonlinearity
- Backpropagation = running stochastic gradient descent through a multilayer network efficiently
- Weight decay = regularization / Bayesian prior smoothing

Word Representations

The standard word representation

The vast majority of rule-based **and** statistical NLP work regards words as atomic symbols: hotel, conference, walk

In vector space terms, this is a vector with one 1 and a lot of zeroes

[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

Dimensionality: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

We call this a "one-hot" representation. Its problem:

motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0

Distributional similarity based representations

You can get a lot of value by representing a word by means of its neighbors

> "You shall know a word by the company it keeps"
>
> (J. R. Firth 1957: 11)

One of the most successful ideas of modern statistical NLP

government debt problems turning into	banking	crises as has happened in
saying that Europe needs unified	banking	regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

You can vary whether you use local or large context to get a more syntactic or semantic clustering

Two traditional word representations: Class-based and soft clustering

Class based models learn word classes of similar words based on distributional information (~ class HMM)

- Brown clustering (Brown et al. 1992, Liang 2005)
- Exchange clustering (Martin et al. 1998, Clark 2003)

Soft clustering models learn for each cluster/topic a distribution over words of how likely that word is in each cluster

- Latent Semantic Analysis (LSA/LSI), Random projections
- Latent Dirichlet Analysis (LDA), HMM clustering

Neural word embeddings as a distributed representation

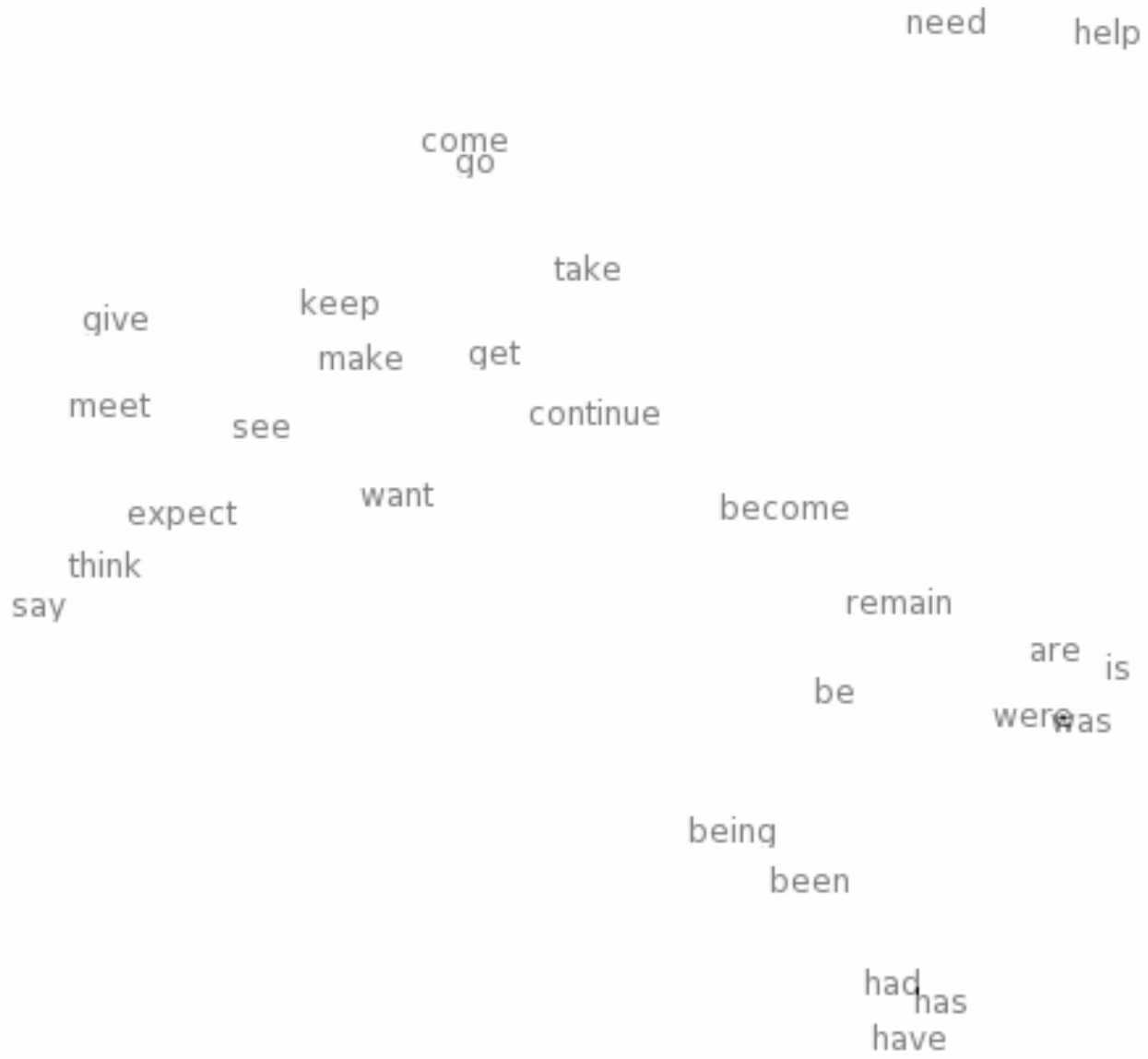
Similar idea

Combine vector space semantics with the prediction of probabilistic models (Bengio et al. 2003, Collobert & Weston 2008, Huang et al. 2012)

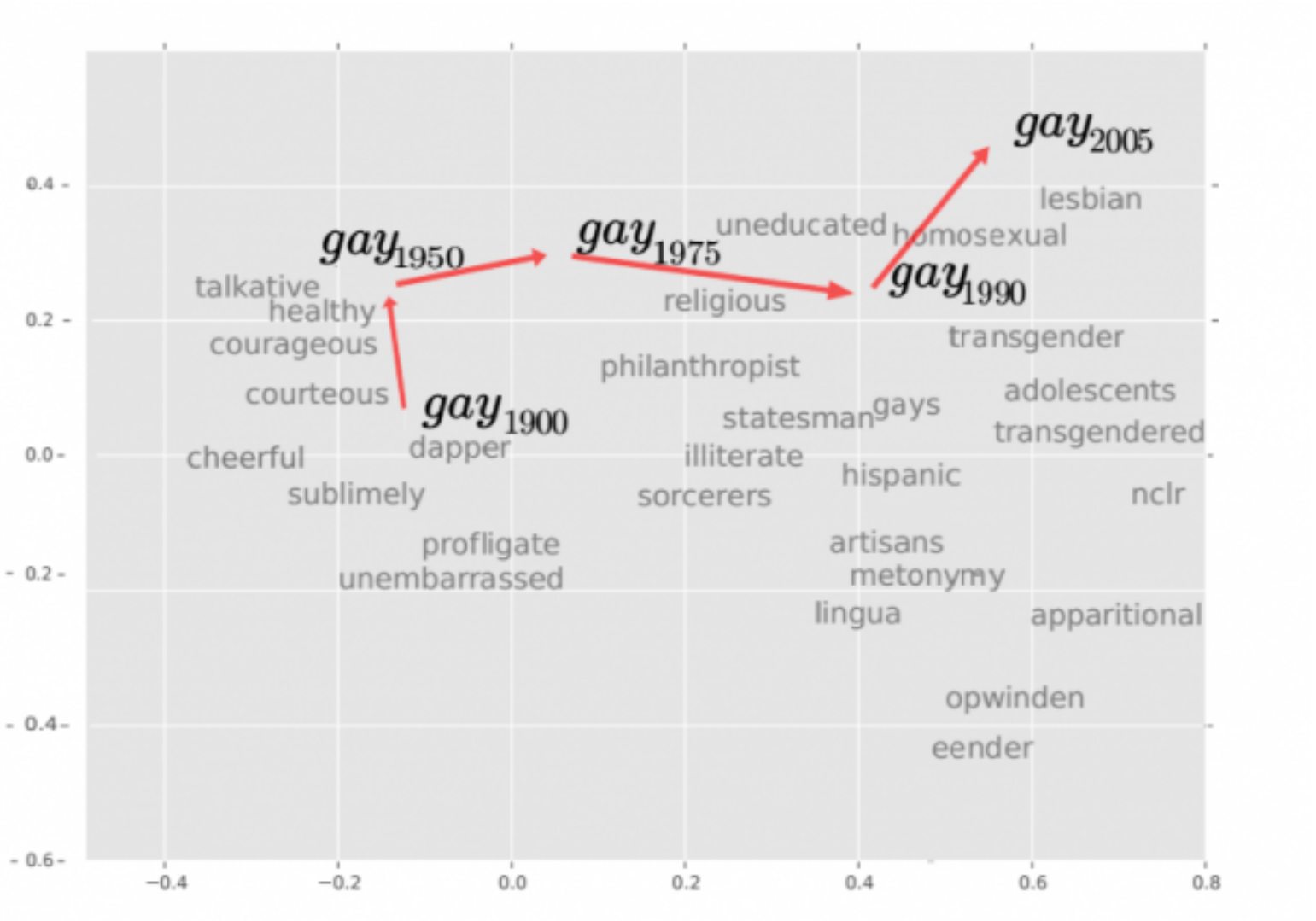
In all of these approaches, including deep learning models, a word is represented as a dense vector

linguistics = $\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$

Neural word embeddings - visualization



Kulkarni, Al-Rfou, Perozzi, & Skiena (2015)



Advantages of the neural word embedding approach

Compared to other methods, neural word embeddings can become more meaningful through adding supervision from one or multiple tasks

- For instance, sentiment is usually not captured in unsupervised word embeddings but can be in neural word vectors

We can build compositional vector representations for longer phrases (next lecture)